Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning
Special Session on KR and Machine Learning

Faithful Approaches to Rule Learning

David J. Tena Cucala , Bernardo Cuenca Grau , Boris Motik
University of Oxford
{ david.tenacucala, bernardo.grau, boris.motik }@cs.ox.ac.uk

Abstract

Rule learning involves developing machine learning models that can be applied to a set of logical facts to predict additional facts, as well as providing methods for extracting from the learned model a set of logical rules that explain symbolically the model's predictions. Existing such approaches, however, do not describe formally the relationship between the model's predictions and the derivations of the extracted rules; rather, it is often claimed without justification that the extracted rules 'approximate' or 'explain' the model, and rule quality is evaluated by manual inspection. In this paper, we study the formal properties of Neural-LP—a prominent rule learning approach. We show that the rules extracted from Neural-LP models can be both unsound and incomplete: on the same input dataset, the extracted rules can derive facts not predicted by the model, and the model can make predictions not derived by the extracted rules. We also propose a modification to the Neural-LP model that ensures that the extracted rules are always sound and complete. Finally, we show that, on several prominent benchmarks, the classification performance of our modified model is comparable to that of the standard Neural-LP model. Thus, *faithful* learning of rules is feasible from both a theoretical and practical point of view.

1 Introduction

Datalog (Abiteboul, Hull, and Vianu 1995) is a widely-used formalism that can represent 'if-then' *rules*. Datalog rules are iteratively applied to a set of facts to derive fresh facts until a fixpoint is reached, allowing Datalog to express important second-order properties such as transitive closure. Due to its expressive power, Datalog is often used as a knowledge representation formalism in the AI community (Eisner and Filardo 2010), and it is seen as the quintessential recursive query language in the database community (Arni et al. 2003; Green et al. 2013).

As with most logic-based KR languages, a key hindrance in practice is that Datalog rules typically need to be constructed manually by experts familiar with both the application domain and the principles of logical deduction. Such expertise is often scarce, and the rule construction process is laborious and costly. Thus, numerous approaches have been proposed with the aim of automating rule construction.

Rule mining approaches analyse a set of facts to identify patterns (e.g., paths of connected facts) and generalise them to rules (Galárraga et al. 2015; Meilicke et al. 2019; Ahmadi et al. 2020). They are typically based on heuristics and do not provide any formal guarantees about their output.

Inductive Logic Programming (ILP) techniques take as input positive and negative examples of inferences, and output a set of rules that can reproduce all such inferences (Muggleton 1991; Cropper et al. 2021). Standard ILP techniques, however, satisfy all examples *exactly* and, as a result, are intolerant to noise: small inconsistencies in the input examples can significantly affect the quality of the produced rules (Evans and Grefenstette 2018).

The approaches in the next group define a transformation $T_{\mathcal{M}}$ of datasets by introducing an ML model $\mathcal{M}$, showing how to encode a set of facts so that $\mathcal{M}$ can be applied, and describing how the model's output is decoded back into a set of facts. The objective is to learn $\mathcal{M}$ so that transformation $T_{\mathcal{M}}$ satisfies the training examples *optimally* (rather than exactly) and thus increase resilience to noise in the training data. Model $\mathcal{M}$ is usually designed to simulate aspects of logical reasoning. For example, the immediate consequence operator of certain classes of logic programs can be approximated by recurrent (Hölldobler, Kalinke, and Störr 1999), fibring (Bader, d'Avila Garcez, and Hitzler 2005), and feed-forward networks (Bader et al. 2007). Also, certain neural-symbolic architectures can simulate the intuition of forward (Dong et al. 2019; Campero et al. 2018) and backwards chaining (Rocktäschel and Riedel 2017) of Horn clauses. Although these techniques are sometimes described as *rule learning* approaches, they do not provide a way to extract rules from the model $\mathcal{M}$. Thus, operator $T_{\mathcal{M}}$ is a 'black box' that aims to mimic Datalog reasoning, but whose inferences cannot be explained symbolically.

Finally, *rule learning* approaches follow the principles outlined in the previous paragraph, but they also specify how to extract a set of Datalog rules $\mathcal{R}_{\mathcal{M}}$ from $\mathcal{M}$. Neural-LP (Yang, Yang, and Cohen 2017) is a prominent example that has been very influential in the field, and it has spurred the development of several refinements (Sadeghian et al. 2019) and extensions to more complex rules (Wang et al. 2020). The ∂ILP (Evans and Grefenstette 2018) and RNNLogic (Qu et al. 2021) frameworks follow the same principle, but use a different kind of ML model.

The rule learning approaches mentioned in the previous paragraph promise to deliver the best of both symbolic reasoning and machine learning. To the best of our knowledge,

however, these approaches do not provide any formal guarantees on the relationship between $\mathcal{R}_{\mathcal{M}}$ and $T_{\mathcal{M}}$. In particular, $\mathcal{R}_{\mathcal{M}}$ should at least be a *sound* approximation of $T_{\mathcal{M}}$, in the sense that, for an arbitrary set of facts D, all the facts obtained by applying $\mathcal{R}_{\mathcal{M}}$ to D should be contained in the result of applying $T_{\mathcal{M}}$ to D. Without this property, the extent to which the extracted rules explain the model's predictions is inherently unclear. Ideally, though, $\mathcal{R}_{\mathcal{M}}$ should represent $\mathcal{M}$ *faithfully*: applying either $T_{\mathcal{M}}$ or $\mathcal{R}_{\mathcal{M}}$ to an arbitrary set of facts should produce *exactly* the same results. Surprisingly, this issue has been overlooked in the literature: it is often claimed without justification that the extracted rules 'approximate' or 'explain' the model, and rule quality is usually evaluated just by manual inspection.

In this paper, we make the first steps towards studying formally the rule extraction procedures proposed in the literature. We focus on Neural-LP since, on the one hand, Yang, Yang, and Cohen (2017) present explicitly the rule extraction algorithm for their approach, and, on the other hand, Neural-LP has been successfully trained on large inputs and has inspired many other approaches (Sadeghian et al. 2019; Wang et al. 2020). While our results (naturally) depend on the specifics of the Neural-LP model, we believe they can be adapted to other rule learning approaches.

After recapitulating the definitions of Neural-LP in Section 2, we analyse the formal properties of Neural-LP models and the rule extraction algorithm in Section 3. To this end, we first reformulate the definitions of Neural-LP and discuss the underlying intuitions. The extraction procedure takes as input a threshold parameter. We then show that, for arbitrary thresholds, the extracted rules can be unsound (i.e., on some datasets, the rules can derive facts not predicted by the model); however, we can always ensure soundness by selecting the threshold in a specific way. In contrast, we prove that it is *impossible* to devise a procedure that takes an arbitrary Neural-LP model and produces rules that faithfully represent the model's predictions. Intuitively, predictions of Neural-LP models can depend on *how many times* a rule body matches the data, whereas inferences in standard Datalog can only depend on whether there is at least one way to match a rule body to the data.

As a possible remedy, in Section 4 we propose a new family of *max-Neural-LP models* for which the rule extraction procedure by Yang, Yang, and Cohen (2017) always produces faithful rules. Intuitively, our proposal prevents models from making predictions based on the number of matches of a rule body, which in turn reduces the expressive power of models to fit within that of standard Datalog and thus makes the extraction of faithful rules theoretically feasible.

Rule learning approaches have commonly been evaluated on knowledge graph completion tasks in the literature (Rossi et al. 2021; Bordes et al. 2013; Teru, Denis, and Hamilton 2020). Following this practice, we evaluated our techniques using several benchmarks in this area, and we present our results in Section 5. We first show that the rules extracted from Neural-LP models do not accurately approximate the models in practice: on our benchmarks, the extracted rules derive only a fraction of the facts predicted by the trained models. Thus, the extracted rules are insufficient to explain or characterise the transformation induced by the models. We then compare the classification performance of Neural-LP and our max-Neural-LP models. Our results show that the proposed modifications do not impact the model's practical ability to make useful predictions: on all benchmarks, max-Neural-LP models can be effectively trained in practice to achieve performance similar to that of standard Neural-LP. We thus show that effective rule learning can be realised in practice without sacrificing faithfulness of rule extraction.

2 Background

We now recapitulate the definitions of Datalog, introduce the Neural-LP model, and discuss its rule extraction algorithm.

Datalog assumes a *signature* consisting of *constants* and *predicates*, where each predicate is associated with a non-negative integer *arity*. A *term* is a *variable* or a constant; an *atom* is an expression of the form $R(t_1, \dots, t_n)$ where R is an n-ary predicate and each t_i, $1 \leq i \leq n$, is a term; a *fact* is a variable-free atom; and a *dataset* is a finite set of facts. A (Datalog) *rule* is an expression of form (1), where $n \geq 0$, H is a *head* atom, and each B_i with $1 \leq i \leq n$ is a *body* atom.

$$H \leftarrow B_1 \wedge \cdots \wedge B_n \tag{1}$$

A (Datalog) *program* is a finite set of rules. We use (possibly indexed) letters $a, b, c \dots$ for constants, letters $x, y, z \dots$ for variables, and letters $R, S, T, \dots$ for predicates.

A *substitution* σ is a partial mapping of variables to constants with a finite domain. For α a term, atom, rule, or a set of rules, $\alpha\sigma$ is the result of replacing each occurrence of a variable x in α with $\sigma(x)$, provided that σ is defined on x.

Each rule r of the form (1) gives rise to the *immediate consequence* operator T_r, whose application to a dataset D is defined as the smallest dataset $T_r(D)$ containing $H\sigma$ for each substitution σ of the variables of r to constants of D that satisfies $\{B_1\sigma, \dots, B_n\sigma\} \subseteq D$. For $\mathcal{R}$ a program, let $T_{\mathcal{R}}(D) = \bigcup_{r \in \mathcal{R}} T_r(D)$. Operator $T_{\mathcal{R}}$ is *monotonic*: for all datasets D_1 and D_2, if $D_1 \subseteq D_2$, then $T_{\mathcal{R}}(D_1) \subseteq T_{\mathcal{R}}(D_2)$. Thus, for each dataset D, operator $T_{\mathcal{R}}$ has a unique least fixpoint $T^{\infty}_{\mathcal{R}}(D)$; this set, as well as the process of computing it, are often called the *materialisation* of $\mathcal{R}$ on D.

In our work, we allow rules to be *unsafe* (i.e., variables may occur in the rule head without also occurring in the rule body), and we allow rule bodies to be empty. These assumptions allow us to present the definitions of Neural-LP more seamlessly, but they do not affect the fundamental nature of Datalog. In particular, operator T_r remains well defined; for example, if r is $R(x, y) \leftarrow$ and $D = \{S_1(a), S_2(b)\}$, then $T_r(D) = \{R(a, a), R(a, b), R(b, a), R(b, b)\}$.

For μ a mapping of constants to constants and D a dataset, $\mu(D)$ is the dataset obtained from D by replacing each occurrence of a constant c in D with $\mu(c)$ (and eliminating any duplicate facts so that $\mu(D)$ is a set). It is well known that each Datalog program $\mathcal{R}$ is *invariant* under any μ that maps each constant occurring in $\mathcal{R}$ to itself—that is, for any such μ, we have $\mu(T_{\mathcal{R}}(D)) \subseteq T_{\mathcal{R}}(\mu(D))$ for each dataset D.

Vectors, Matrices, and Tensors. We use tensors over the set $\mathbb{R}$ of real numbers. For $\mathbf{A}$ a tensor with n dimensions, $a_{i_1, i_2, \dots, i_n}$ is its element in position i_1 of the first dimension,

position i_2 of the second dimension, etc. We consider vectors and matrices as tensors with one and two dimensions, respectively. Finally, $\mathbf{v}^\intercal$ is the transpose of a vector $\mathbf{v}$.

Neural-LP assumes a signature consisting of a finite number δ of *binary* predicates arranged in an arbitrary (but fixed) sequence $R_1, \dots, R_\delta$. Each dataset or rule can mention only the predicates given in the signature, so fixing an enumeration of the predicates allows us to represent an arbitrary dataset as an adjacency tensor.

The objective of Neural-LP is to learn an operator $T_\mathcal{M}$ that mimics the immediate consequence operator of a set of *chain rules* of the form (2), where $n \geq 0$.

$$R_h(x_n, x_0) \leftarrow R_{k_n}(x_n, x_{n-1}) \wedge \cdots \wedge R_{k_1}(x_1, x_0) \quad (2)$$

For $n = 0$, rule (2) reduces to $R_h(x_0, x_0) \leftarrow$; when applied to a dataset D, this rule derives $R_h(a, a)$ for each constant a in D. Instead of listing the rules explicitly, $T_\mathcal{M}$ is defined by a machine learning model $\mathcal{M}$. We next define $\mathcal{M}$ and $T_\mathcal{M}$, and we leave the analysis of the model's properties and the discussion of the intuitions behind it to Section 3.

A Neural-LP model $\mathcal{M}$ is a triple $(\mathbf{A}, \mathbf{B}, \beta)$ where, for a positive integer L called the *depth* of the model, $\mathbf{A}$ is a tensor of dimension $\delta \times \delta \times L$, $\mathbf{B}$ is a tensor of dimension $\delta \times (L+1) \times (L+1)$, and $\beta \in \mathbb{R}$. The elements of tensors $\mathbf{A}$ and $\mathbf{B}$ are learnable, whereas β is a classification threshold that is typically given explicitly. The formal description of Neural-LP does not impose any restriction on the range of the components of $\mathbf{A}$ and $\mathbf{B}$; however, the Neural-LP implementation computes $\mathbf{A}$ and $\mathbf{B}$ using the softmax function, so their elements have values between zero and one.

We next discuss how to compute $T_\mathcal{M}(D)$ for a dataset D. First, we identify the number ϵ of distinct constants in D, we order all constants into an arbitrary sequence $c_1, \dots, c_\epsilon$, and we construct an adjacency tensor $\mathbf{M}$ of dimension $\delta \times \epsilon \times \epsilon$ such that $m_{k,i,j}$ is set to 1 if $R_k(c_i, c_j) \in D$ and to 0 otherwise. For k with $1 \leq k \leq \delta$, we let $\mathbf{M}_k$ be the matrix of dimension $\epsilon \times \epsilon$ where $(m_k)_{i,j} = m_{k,i,j}$. Second, for each fact of the form $R_h(c_i, c_j)$ with $1 \leq h \leq \delta$ and $1 \leq i, j \leq \epsilon$, we determine whether $R_h(c_i, c_j)$ should be added to $T_\mathcal{M}(D)$ as follows.

- We define a vector $\mathbf{u}^0$ of dimension ϵ such that $u^0_j = 1$ and all other components are zero.
- For each $\ell \in \{1, \dots, L\}$, we inductively compute vectors $\mathbf{u}^\ell$ using the following formula:

$$\mathbf{u}^\ell = \sum_{k=1}^{\delta} a_{h,k,\ell} \cdot \mathbf{M}_k \times \left(\sum_{\ell'=0}^{\ell-1} b_{h,\ell,\ell'+1} \cdot (\mathbf{u}^{\ell'})^\intercal \right). \quad (3)$$

- We compute vector $\mathbf{u}^{L+1}$ as follows:

$$\mathbf{u}^{L+1} = \sum_{\ell=0}^{L} b_{h,L+1,\ell+1} \cdot \mathbf{u}^\ell. \quad (4)$$

- We add $R_h(c_i, c_j)$ to $T_\mathcal{M}(D)$ if and only if $u^{L+1}_i > \beta$.

Neural-LP also provides a procedure for extracting a set of rules from a model $\mathcal{M}$, which we show in Algorithm 1. The procedure is parameterised by a threshold γ (which can be different from the model's classification threshold β). Since the aim is to learn only chain rules, the body of each such rule is represented in the extraction algorithm as a possibly empty sequence of predicates $[R_{k_\ell}, \dots, R_{k_1}]$; we use $[]$ as the empty sequence and $[\rho|R]$ as the concatenation of sequence ρ with predicate R. The algorithm considers each predicate R_h in the signature and produces in lines 2–15 all chain rules of depth up to L with R_h in the head. To this end, the algorithm computes a sequence $\mathcal{R}_0, \dots, \mathcal{R}_{L+1}$ of pairs of the form $\langle s, \rho \rangle$, where ρ is a sequence corresponding to a subset of the rule body and s is a real-valued confidence factor; we explain the intuition behind this step in more detail in Section 3. Finally, the algorithm selects each rule body whose confidence exceeds the threshold γ in line 14, and it converts it into a rule with R_h in the head in line 15.

Algorithm 1 Neural-LP's Rule Extraction Algorithm

Inputs: a Neural-LP model $\mathcal{M} = (\mathbf{A}, \mathbf{B}, \beta)$ of depth L, a threshold $\gamma \in \mathbb{R}$
Output: a Datalog program

1: $\mathcal{R} := \emptyset$
2: **for** $h \in \{1, \dots, \delta\}$ **do**
3: $\quad \mathcal{R}_0 := \{\langle 1, [] \rangle\}$
4: $\quad$ **for** $\ell \in \{1, \dots, L+1\}$ **do**
5: $\quad\quad \hat{\mathcal{R}}_\ell := \emptyset$
6: $\quad\quad$ **for** $\ell' \in \{1, \dots, \ell\}$ and $\langle s, \rho \rangle \in \mathcal{R}_{\ell'-1}$ **do**
7: $\quad\quad\quad$ add $\langle s \cdot b_{h,\ell,\ell'}\ ,\ \rho \rangle$ to $\hat{\mathcal{R}}_\ell$
8: $\quad\quad$ **if** $\ell \leq L$ **then**
9: $\quad\quad\quad \mathcal{R}_\ell := \emptyset$
10: $\quad\quad\quad$ **for** $\langle s, \rho \rangle \in \hat{\mathcal{R}}_\ell$ and $k \in \{1, \dots, \delta\}$ **do**
11: $\quad\quad\quad\quad$ add $\langle s \cdot a_{h,k,\ell}\ ,\ [\rho|R_k] \rangle$ to $\mathcal{R}_\ell$
12: $\quad\quad$ **else**
13: $\quad\quad\quad \mathcal{R}_\ell := \hat{\mathcal{R}}_\ell$
14: $\quad$ **for** $\langle s, [R_{k_n}, \dots, R_{k_1}] \rangle \in \mathcal{R}_{L+1}$ such that $s > \gamma$ **do**
15: $\quad\quad$ add rule (2) to $\mathcal{R}$
16: **return** $\mathcal{R}$

3 Analysing the Neural-LP Model

In this section we investigate the formal properties of the Neural-LP approach. In particular, in Section 3.1 we describe intuitively the inferences that a Neural-LP model can make, and we reformulate equations (3) and (4) in a way that allows us to draw additional insights and prove our technical results more easily. In Section 3.2, we present a succinct characterisation of the Datalog programs produced by Algorithm 1. Finally, in Section 3.3 we discuss how the predictions of a Neural-LP model and the derivations of the rules extracted from the model relate on arbitrary datasets.

3.1 Analysing the Neural-LP Model

Yang, Yang, and Cohen (2017) motivate their formulation of Neural-LP by the desire to obtain a trainable model that mimics the application of chain rules of varying length. The definitions of Neural-LP can perhaps be better understood if we observe that the model does not simulate chain rules directly; rather, it simulates a nonstandard application of rules

of the form (5)–(7) instantiated for all h, k, ℓ, and ℓ' satisfying $1 \leq h, k \leq \delta$ and $0 \leq \ell' < \ell \leq L$.

$$U_{h,0}(x, x) \leftarrow \tag{5}$$

$$U_{h,\ell}(x, z) \leftarrow R_k(x, y) \wedge U_{h,\ell'}(y, z) \tag{6}$$

$$U_{h,L+1}(x, y) \leftarrow U_{h,\ell}(x, y). \tag{7}$$

Here, $U_{h,\ell}$ are auxiliary binary predicates distinct from the signature predicates $R_1, \ldots, R_\delta$; we assume that $U_{h,\ell}$ do not occur in any input dataset, so we consider them to be 'internal' to the model. A model $\mathcal{M} = (\mathbf{A}, \mathbf{B}, \beta)$ assigns a weight to each body atom in the aforementioned rules:

- in each rule (6), element $a_{h,k,\ell}$ of $\mathbf{A}$ determines the weight of atom $R_k(x, y)$, and element $b_{h,\ell,\ell'+1}$ of $\mathbf{B}$ determines the weight of atom $U_{h,\ell'}(x, z)$; and
- in each rule (7), element $b_{h,L+1,\ell+1}$ of $\mathbf{B}$ determines the weight of atom $U_{h,\ell}(x, y)$.

Note that elements $b_{h,\ell,\ell'+1}$ of $\mathbf{B}$ with $\ell' \geq \ell$ are not used in the predictions made by $\mathcal{M}$.

The application of model $\mathcal{M}$ to a dataset can then be seen as applying rules (5)–(7) in the following nonstandard way. Assume that each fact in the dataset is annotated with a confidence value; for uniformity, we define the confidence value of all facts not contained in the dataset as zero. An application of a rule *increases* the confidence value of the derived fact by the product of the weights of all body atoms and the confidence values of the matched facts; for rules with empty bodies, we define this product to be one.

We can now reinterpret equations (3) and (4) as applying rules (5)–(7) with weighted atoms. In particular, let D be an arbitrary dataset and let D' be obtained by applying rules (5)–(7) as outlined in the previous paragraph, but where atoms $R_k(x, y)$ are only matched to facts in D with confidence value of one. Finally, assume that, to check whether $R_h(c_i, c_j)$ should be added to $T_\mathcal{M}(D)$, we compute vectors $\mathbf{u}^\ell$ as specified in Section 2. Then, for each $0 \leq \ell \leq L + 1$ and $1 \leq p \leq \epsilon$, the confidence of fact $U_{h,\ell}(c_p, c_j)$ in D' is equal to u_p^ℓ; moreover, $R_h(c_i, c_j) \in T_\mathcal{M}(D)$ if and only if the confidence of $U_{h,L+1}(c_i, c_j)$ is larger than β. Indeed, for $\ell = 0$, facts of the form $U_{h,0}(c_p, c_j)$ are derived only by (5), so the confidence of each fact is $u_p^0 = 1$ if $p = j$ and $u_p^0 = 0$ if $p \neq j$. For $1 \leq \ell \leq L$, equation (3) can be rewritten as

$$\mathbf{u}^\ell = \sum_{k=1}^{\delta} \sum_{\ell'=0}^{\ell-1} a_{h,k,\ell} \cdot b_{h,\ell,\ell'+1} \cdot \mathbf{M}_k \times (\mathbf{u}^{\ell'})^\intercal, \tag{8}$$

which clearly corresponds to the computations of rules (6). Finally, equation (4) corresponds to applying rules (7).

Chain rules of varying length can be obtained by *unfolding* the auxiliary predicates. For example, we can eliminate predicate $U_{h,1}$ in rules $U_{h,2}(x, z) \leftarrow R_2(x, y) \wedge U_{h,1}(y, z)$ and $U_{h,1}(x, z) \leftarrow R_3(x, y) \wedge U_{h,0}(y, z)$ by unifying and then resolving away the atoms containing $U_{h,1}$; we thus obtain $U_{h,2}(x, z) \leftarrow R_2(x, y) \wedge R_3(y, w) \wedge U_{h,0}(w, z)$ where w is a fresh variable. It is straightforward to see that, if we extend rules (5)–(7) with rules

$$R_h(x, y) \leftarrow U_{h,L+1}(x, y) \text{ for } h \in \{1, \ldots, \delta\} \tag{9}$$

and then unfold all auxiliary predicates, we obtain all chain rules of the form (2) with $0 \leq n \leq L$.

We next formalise the intuitions of such unfolding in a way that will facilitate the proofs of our main results. Our reformulation uses (possibly empty) sequences of integers that we will use as indexes into the model's tensors.

- For each $s \in \{0, \ldots, L\}$, let $\mathcal{L}_s$ be the set of all nonempty integer sequences $[\ell_0, \ldots, \ell_n]$ such that $0 \leq n \leq s$ and $0 = \ell_0 < \ell_1 < \cdots < \ell_n = s$. Note that $\mathcal{L}_0 = \{[0]\}$.
- Let $\mathcal{L} = \bigcup_{0 \leq s \leq L} \mathcal{L}_s$.
- For $\lambda = [\ell_0, \ldots, \ell_n] \in \mathcal{L}$, let $\mathcal{K}_\lambda$ be the set of all sequences $[k_1, \ldots, k_n]$ where each k_p satisfies $1 \leq k_p \leq \delta$.
- For $\lambda = [\ell_0, \ldots, \ell_n] \in \mathcal{L}$ and $i, j \in \{1, \ldots, \epsilon\}$, let $\mathcal{T}_{\lambda,i,j}$ be the set of all sequences $[t_0, \ldots, t_n]$ where $t_0 = j$, $t_n = i$, and each t_p satisfies $1 \leq t_p \leq \epsilon$.

Note that all sequences are nonempty apart from the only sequence in $\mathcal{K}_{[0]} = \{[]\}$. Furthermore, for $h \in \{1, \ldots, \delta\}$ and $\lambda = [\ell_0, \ldots, \ell_n]$, $\kappa = [k_1, \ldots, k_n]$, and $\tau = [t_0, \ldots, t_n]$, let

$$\varphi(h, \kappa, \lambda) = \prod_{i=1}^{n} a_{h,k_i,\ell_i} \cdot b_{h,\ell_i,\ell_{i-1}+1},$$

$$\psi(\kappa, \tau) = \prod_{i=1}^{n} m_{k_i,t_i,t_{i-1}}, \qquad \text{and}$$

$$\theta(h, \kappa, \lambda) = b_{h,L+1,\ell_n+1} \cdot \varphi(h, \kappa, \lambda).$$

With these definitions in place, we can now show that equations (3) and (4) are equivalent to (10) and (11), respectively.

Proposition 1. *Let $\mathcal{M} = (\mathbf{A}, \mathbf{B}, \beta)$ be a Neural-LP model, let D be a dataset, and let $\mathbf{u}^0, \ldots, \mathbf{u}^{L+1}$ be the vectors computed by equations* (3) *and* (4) *when checking whether fact $R_h(c_i, c_j)$ should be added to $T_\mathcal{M}(D)$. Then,* (10) *and* (11) *hold for each $0 \leq \ell \leq L$ and each $1 \leq p \leq \epsilon$.*

$$u_p^\ell = \sum_{\lambda \in \mathcal{L}_\ell} \sum_{\kappa \in \mathcal{K}_\lambda} \varphi(h, \kappa, \lambda) \sum_{\tau \in \mathcal{T}_{\lambda,p,j}} \psi(\kappa, \tau) \tag{10}$$

$$u_p^{L+1} = \sum_{\lambda \in \mathcal{L}} \sum_{\kappa \in \mathcal{K}_\lambda} \theta(h, \kappa, \lambda) \sum_{\tau \in \mathcal{T}_{\lambda,p,j}} \psi(\kappa, \tau) \tag{11}$$

Proof. We prove (10) by induction on ℓ. Consider $\ell = 0$. Then, we have $\mathcal{L}_0 = \{\lambda\}$ for $\lambda = [0]$, and $\mathcal{K}_\lambda = \{\kappa\}$ where $\kappa = []$; in addition, $\varphi(h, \kappa, \lambda) = 1$. Furthermore, if $p \neq j$, then $\mathcal{T}_{\lambda,p,j} = \emptyset$ and so $u_p^\ell = 0$. Otherwise, $\mathcal{T}_{\lambda,j,j} = \{[j]\}$, so $\sum_{\tau \in \mathcal{T}_{\lambda,j,j}} \psi(\kappa, \tau) = 1$ and we have $u_j^\ell = 1$.

Now assume that (10) holds for $\ell - 1$ and consider some u_p^ℓ computed by (8). The formula for u_p^ℓ can be rewritten as in Figure 1 using the distributive properties of multiplication and summation. In the last step, note that $\lambda \in \mathcal{L}_{\ell'}$ extended with ℓ produces a sequence $\lambda' \in \mathcal{L}_\ell$, extending κ with k produces κ', and extending τ with t produces τ'. The proof of formula (11) is analogous and is omitted for brevity. □

Intuitively, these formulas correspond to the 'unfolding' of rules (5)–(7). Each sequence $\lambda \in \mathcal{L}$ determines a path through the auxiliary body predicates, and each sequence

$$
\begin{aligned}
u_p^\ell &= \sum_{k=1}^{\delta} \sum_{\ell'=0}^{\ell-1} a_{h,k,\ell} \cdot b_{h,\ell,\ell'+1} \left(\sum_{t=1}^{\epsilon} m_{k,p,t} \cdot u_t^{\ell'} \right) = \\
&= \sum_{\ell'=0}^{\ell-1} \sum_{k=1}^{\delta} a_{h,k,\ell} \cdot b_{h,\ell,\ell'+1} \left(\sum_{t=1}^{\epsilon} m_{k,p,t} \sum_{\lambda \in \mathcal{L}_{\ell'}} \sum_{\kappa \in \mathcal{K}_\lambda} \varphi(\kappa, \lambda) \sum_{\tau \in \mathcal{T}_{\lambda,p,j}} \psi(\kappa, \tau) \right) = \\
&= \sum_{\ell'=0}^{\ell-1} \sum_{k=1}^{\delta} a_{h,k,\ell} \cdot b_{h,\ell,\ell'+1} \left(\sum_{\lambda \in \mathcal{L}_{\ell'}} \sum_{\kappa \in \mathcal{K}_\lambda} \varphi(\kappa, \lambda) \sum_{b=1}^{\epsilon} \sum_{\tau \in \mathcal{T}_{\lambda,p,j}} m_{k,p,t} \cdot \psi(\kappa, \tau) \right) = \\
&= \sum_{\ell'=0}^{\ell-1} \sum_{\lambda \in \mathcal{L}_{\ell'}} \sum_{k=1}^{\delta} \sum_{\kappa \in \mathcal{K}_\lambda} a_{h,k,\ell} \cdot b_{h,\ell,\ell'+1} \cdot \varphi(\kappa, \lambda) \sum_{t=1}^{\epsilon} \sum_{\tau \in \mathcal{T}_{\lambda,p,j}} m_{k,p,t} \cdot \psi(\kappa, \tau) = \sum_{\lambda' \in \mathcal{L}_\ell} \sum_{\kappa' \in \mathcal{K}_{\lambda'}} \varphi(\kappa', \lambda') \sum_{\tau' \in \mathcal{T}_{\lambda',p,j}} \psi(\kappa', \tau')
\end{aligned}
$$

Figure 1: Equivalences in the Proof of Proposition 1

$\kappa \in \mathcal{K}_\lambda$ identifies the body predicates associated to this path. In particular, each ℓ_i in λ and k_i in κ together identify a rule of the form (6) for which $\ell = \ell_i$, $\ell' = \ell_{i-1}$, and $k = k_i$. Finally, each sequence $\tau \in \mathcal{T}_{\lambda,p,j}$ identifies a possible set of facts $R_{k_i}(c_{t_i}, c_{t_{i-1}})$ that can potentially be matched to the body atoms of the unfolded rule, and elements $m_{k_i,t_i,t_{i-1}}$ determine whether these facts are contained in D. Thus, equations (10) and (11) can indeed be understood as applications of the unfolding of rules (5)–(7) to D.

3.2 Characterisation of Rule Extraction

This idea of unfolding is also reflected in the rule extraction procedure shown in Algorithm 1. In particular, when the loop in lines 2–15 completes for some h, all unfoldings having predicate $U_{h,\ell}$, $0 \leq \ell \leq L$, in the head are described by $\mathcal{R}_\ell$, and all unfoldings having predicate R_h in the head are described by $\mathcal{R}_{L+1}$. Specifically, for each $\langle s, \rho \rangle$ in one of these sets, s corresponds to the product of the weights of all body atoms in the unfolding, and ρ describes the body of the unfolding. This claim can be easily proved by induction on ℓ. For $\ell = 0$, the only unfolding with $U_{h,0}$ in the head is $U_{h,0}(x, x) \leftarrow$ and the product of all body weights is 1, which matches the definition of $\mathcal{R}_0$ in line 3. For $1 \leq \ell \leq L$, the induction hypothesis and lines 5–7 ensure that, for each $\langle s, \rho \rangle \in \hat{\mathcal{R}}_\ell$, sequence ρ describes the body of an unfolding with some $U_{h,\ell'}$ in the head, where $0 \leq \ell' < \ell$, and s is the product of the body weights of that unfolding and the weight of atom $U_{h,\ell'}(y, z)$ in a rule of form (6); but then, lines 9–11 clearly extend each such $\langle s, \rho \rangle$ to the relevant unfoldings with $U_{h,\ell}$ in the head. Finally, line 13 ensures that $\mathcal{R}_{L+1}$ captures the unfoldings generated from rules of form (7).

Rules (5)–(7) can often be unfolded in different ways to the same rule. For example, rule $R_5(x, y) \leftarrow R_7(x, y)$ can be obtained by unfolding (7) for $\ell = 3$, (6) for $k = 7$, $\ell = 3$, and $\ell' = 1$, and (5), as well as by unfolding (7) for $\ell = 2$, (6) for $k = 7$, $\ell = 2$, and $\ell' = 1$, and (5). Each such unfolding can derive facts with different confidence factors.

Before proceeding, we present a more direct characterisation of the output of Algorithm 1, which will allow us to formalise other statements about Neural-LP more easily.

Definition 1. *For $\mathcal{M} = (\mathbf{A}, \mathbf{B}, \beta)$ a Neural-LP model and $\gamma \in \mathbb{R}$, program $\mathcal{R}^\gamma_\mathcal{M}$ contains a rule of the form* (2) *for each index $h \in \{1, \ldots, \delta\}$, each sequence $\lambda \in \mathcal{L}$, and each sequence $\kappa = [k_1, \ldots, k_n] \in \mathcal{K}_\lambda$ such that $\theta(h, \kappa, \lambda) > \gamma$.*

Proposition 2. *For each Neural-LP model $\mathcal{M}$ and each $\gamma \in \mathbb{R}$, Algorithm 1 outputs $\mathcal{R}^\gamma_\mathcal{M}$.*

Proof. Consider an arbitrary index $h \in \{1, \ldots, \delta\}$ and let $\mathcal{R}_0, \ldots, \mathcal{R}_{L+1}$ be the sets computed by Algorithm 1 when the iteration of the loop in lines 2–15 finishes for h. We show that, for each $\ell \in \{0, \ldots, L\}$, each sequence of predicates $\rho = [R_{k_n}, \ldots, R_{k_1}]$ with $0 \leq n \leq \ell$, and each $s \in \mathbb{R}$, we have $\langle s, \rho \rangle \in \mathcal{R}_\ell$ if and only if there exists a sequence $\lambda \in \mathcal{L}_\ell$ such that $\varphi(h, \kappa, \lambda) = s$ where $\kappa = [k_1, \ldots, k_n]$. The proof is by an easy induction on ℓ and is omitted. Then, lines 6–7 and line 13 for $L+1$, and the way in which the rules are produced in lines 14–15 ensures the claim for h. □

3.3 Relating Model to Extracted Rules

The rule extraction algorithm of Neural-LP is intuitive, but Yang, Yang, and Cohen (2017) never discussed the extent to which the extracted rules characterise the model's inferences. We study this question in the rest of this section. As we mentioned in Section 2, the Neural-LP implementation uses the softmax function to compute $\mathbf{A}$ and $\mathbf{B}$ so the elements of these tensors are nonnegative; but then, it is also reasonable to require the classification threshold to be nonnegative too. Consequently, we shall restrict our attention to models that are *regular* as per the following definition.

Definition 2. *A Neural-LP model $\mathcal{M} = (\mathbf{A}, \mathbf{B}, \beta)$ is* regular *if $\beta \geq 0$ and no element of tensor $\mathbf{A}$ or $\mathbf{B}$ is negative.*

Before proceeding with our analysis, we introduce several notions that capture possible relationships between a model and the extracted rules.

Definition 3. *Let $\mathcal{M}$ be a Neural-LP model, and let $\mathcal{R}$ be a Datalog program. Then, program $\mathcal{R}$ is* sound *(resp.* complete*) for $\mathcal{M}$ if, for each dataset D, $T_\mathcal{R}(D) \subseteq T_\mathcal{M}(D)$ (resp. $T_\mathcal{R}(D) \supseteq T_\mathcal{M}(D)$). Moreover, $\mathcal{R}$ is* faithful *to $\mathcal{M}$ if $\mathcal{R}$ is both sound and complete for $\mathcal{M}$.*

In other words, a sound program $\mathcal{R}$ can derive only consequences also derived by $\mathcal{M}$, but it may not necessarily derive all consequences of $\mathcal{M}$; in contrast, a complete program derives all consequences of $\mathcal{M}$, but it may also derive consequences that are 'incorrect' according to $\mathcal{M}$. Ideally, we would like $\mathcal{R}$ to be faithful to $\mathcal{M}$ since program $\mathcal{R}$ then provides an exact symbolic account of all inferences that $\mathcal{M}$ can make on an arbitrary dataset.

We are now ready to present two results that relate a regular Neural-LP model $\mathcal{M}$ and the corresponding program $\mathcal{R}^\gamma_\mathcal{M}$. We show that $\mathcal{R}^\gamma_\mathcal{M}$ is sound for $\mathcal{M}$ if we choose $\gamma \geq \beta$, but otherwise $\mathcal{R}^\gamma_\mathcal{M}$ can be unsound for $\mathcal{M}$.

Theorem 1.

1. *For each regular Neural-LP model $\mathcal{M} = (\mathbf{A}, \mathbf{B}, \beta)$ and each $\gamma \geq \beta$, program $\mathcal{R}^\gamma_\mathcal{M}$ is sound for $T_\mathcal{M}$.*
2. *There exists a regular Neural-LP model $\mathcal{M} = (\mathbf{A}, \mathbf{B}, \beta)$ such that, for $\gamma < \beta$, program $\mathcal{R}^\gamma_\mathcal{M}$ is unsound for $\mathcal{M}$.*

Proof of Claim 1. Fix an arbitrary regular Neural-LP model $\mathcal{M} = (\mathbf{A}, \mathbf{B}, \beta)$, real number $\gamma \geq \beta$, dataset D, and fact $R_h(c_i, c_j) \in T_{\mathcal{R}^\gamma_\mathcal{M}}(D)$. Then, there exists a rule $r \in \mathcal{R}^\gamma_\mathcal{M}$ of the form (2) and facts

$$\{R_{k_n}(c_{t_n}, c_{t_{n-1}}), \ldots, R_{k_1}(c_{t_1}, c_{t_0})\} \subseteq D$$

such that applying r to these facts derives $R_h(c_i, c_j)$. Now let $\kappa = [k_1, \ldots, k_n]$ and $\tau = [t_0, \ldots, t_n]$. By Definition 1, there exists a sequence $\lambda \in \mathcal{L}$ such that $\theta(h, \kappa, \lambda) > \gamma$. Note that $\kappa \in \mathcal{K}_\lambda$ and $\tau \in \mathcal{T}_{\lambda,i,j}$; moreover, $m_{j,t_i,t_{i-1}} = 1$ for each $1 \leq i \leq n$, so $\psi(\kappa, \tau) = 1$. Now let $\mathbf{u}^0, \ldots, \mathbf{u}^{L+1}$ be the vectors computed as specified in equations (3) and (4) for checking whether $R_h(c_i, c_j)$ should be added to $T_\mathcal{M}(D)$. By Proposition 1, the value of u_i^{L+1} is given by equation (11); all numbers in this equation are nonnegative because model $\mathcal{M}$ is regular, so $\psi(\kappa, \tau) = 1$ and $\theta(h, \kappa, \lambda) > \gamma \geq \beta$ clearly imply $u_i^{L+1} > \beta$. Therefore, $R_h(c_i, c_j)$ is added to $T_\mathcal{M}(D)$, and our claim holds. □

Proof of Claim 2. Let $\delta = 2$, but note that we can handle larger signatures by padding tensors $\mathbf{A}$ and $\mathbf{B}$ with zeros if needed. Now let $\mathcal{M} = (\mathbf{A}, \mathbf{B}, \beta)$ be the regular Neural-LP model of depth $L = 1$ where $\beta = 0.5$ and tensors $\mathbf{A}$ and $\mathbf{B}$ contain zeros everywhere apart from $a_{1,2,1} = 0.5$ and $b_{1,1,1} = b_{1,2,2} = 1$. Moreover, let $D = \{R_2(a, a)\}$; one can easily check that $T_\mathcal{M}(D) = \emptyset$. However, for each $\gamma < \beta$, by Definition 1 program $\mathcal{R}^\gamma_\mathcal{M}$ contains rule (12).

$$R_1(x, y) \leftarrow R_2(x, y) \tag{12}$$

One can easily check that $T_{\mathcal{R}^\gamma_\mathcal{M}}(D) = \{R_1(a, a)\}$; hence, program $\mathcal{R}^\gamma_\mathcal{M}$ is unsound for $\mathcal{M}$. □

Theorem 1 is quite intuitive. In particular, note that each rule extracted from $\mathcal{M}$ corresponds to an element of the sum in equation (11). Thus, by selecting γ such that $\gamma \geq \beta$, we make sure that the consequences of each extracted rule are also consequences of $\mathcal{M}$. In contrast, if we select γ such that $\gamma < \beta$, then the confidence factors of the facts derived by (5)–(7) may not exceed the threshold β and we run the risk of extracting unsound rules.

Along these lines, one might hope that we can ensure completeness by selecting $\gamma = \beta$; however, we next show that this is not the case. In fact, our result is much stronger: we show that there exists a Neural-LP model whose inferences do not correspond to inferences of any Datalog program. As a result, no algorithm can extract a sound and complete program given an arbitrary regular Neural-LP model.

Theorem 2. *There exists a regular Neural-LP model $\mathcal{M}$ such that no Datalog program $\mathcal{R}$ is faithful to $\mathcal{M}$.*

Proof. Let $\delta = 2$, but note that we can handle larger signatures by padding tensors $\mathbf{A}$ and $\mathbf{B}$ with zeros. Now let $\mathcal{M} = (\mathbf{A}, \mathbf{B}, \beta)$ be the regular Neural-LP model of depth $L = 2$ where $\beta = 0.75$ and tensors $\mathbf{A}$ and $\mathbf{B}$ contain zeros everywhere apart from

$$a_{1,2,1} = 0.5 \qquad \text{and}$$
$$a_{1,2,2} = b_{1,1,1} = b_{1,2,2} = b_{1,3,3} = 1.$$

Now assume that there exists a Datalog program $\mathcal{R}$ that is faithful to $\mathcal{M}$, and let D_1 and D_2 be the following datasets, where constant c does not appear in $\mathcal{R}$.

$$D_1 = \{R_2(a, b), R_2(b, d)\} \tag{13}$$
$$D_2 = D_1 \cup \{R_2(a, c), R_2(c, d)\} \tag{14}$$

It is straightforward to check using the definitions of Neural-LP that $T_\mathcal{M}(D_1) = \emptyset$ and $T_\mathcal{M}(D_2) = \{R_1(a, d)\}$. Since $\mathcal{R}$ is complete for $\mathcal{M}$, we have $T_\mathcal{R}(D_2) = \{R_1(a, d)\}$. Now let μ be the mapping of constants to constants that is the identity everywhere apart from $\mu(c) = b$. Note that $\mu(D_2) = D_1$ and $\mu(T_\mathcal{R}(D_2)) = \{R_1(a, d)\}$. Now constant c does not occur in $\mathcal{R}$ and μ is the identity on all other constants, so program $\mathcal{R}$ is invariant under μ and

$$\{R_1(a, d)\} = \mu(T_\mathcal{R}(D_2)) \subseteq T_\mathcal{R}(\mu(D_2)) = T_\mathcal{R}(D_1)$$

holds. But then, $R_1(a, d) \in T_\mathcal{R}(D_1)$ implies that $\mathcal{R}$ is unsound for $\mathcal{M}$, which is a contradiction. □

Theorem 2 can be understood as follows. Model $\mathcal{M}$ used in the proof describes rules whose unfolding corresponds to

$$R_1(x, z) \leftarrow R_2(x, y) \wedge R_2(y, z) \tag{15}$$

where the weights of atoms $R_2(x, y)$ and $R_2(y, z)$ are 1 and 0.5, respectively. Now the body of rule (15) can be matched just once in dataset D_1, so $R_1(a, d)$ is derived with confidence factor 0.5, which does not exceed the classification threshold $\beta = 0.75$. In contrast, the body of (15) matches twice in dataset D_2, so $R_1(a, d)$ is derived with confidence factor 1. In other words, the derivations of Neural-LP can count how many times a rule body matches to a dataset. Indeed, this is shown in equation (11): each κ corresponds to a chain rule of the form (2), and $\psi(\kappa, \tau) = 1$ if the rule matches to the facts identified by the sequence τ; thus, $\sum_{\tau \in \mathcal{T}_{\lambda,p,j}} \psi(\kappa, \tau)$ gives us the number of body matches. Counting the matches of a rule body is not possible in standard Datalog: each Datalog rule can only determine whether the rule body matches or not. Such inferences can be captured only using nonstandard extensions of Datalog such as arithmetic and aggregate functions, but these drastically change the computational properties of Datalog.

4 Ensuring Faithfulness of Rule Extraction

Theorems 1 and 2 suggest that any rule extraction procedure for Neural-LP faces fundamental limitations: the best we can hope for is soundness. One may also wonder whether program $\mathcal{R}^{\beta}_{\mathcal{M}}$ is 'sufficiently complete' in practice in the sense that it derives *most* consequences of a model $\mathcal{M}$. We show empirically in Section 5 that this is not the case: the number of facts derived by $\mathcal{R}^{\beta}_{\mathcal{M}}$ and $T_{\mathcal{M}}$ can be different by several orders of magnitude on commonly used benchmarks. These observations motivate the question of whether the Neural-LP model can be modified in a way that makes extracting faithful Datalog programs possible.

We answer this question positively: we propose a new family of *max-Neural-LP models* for which Algorithm 1 always produces faithful Datalog programs. Intuitively, our modification allows us to replace the sums in equations (10) and (11) with maximum operators. This prevents the model from making predictions based on how many times a rule body matches to the data, which in turn reduces the expressive power of the model to fit that of standard Datalog.

Towards this goal, we note that the sum over sequences τ in equations (10) and (11) actually originates from the matrix product in equation (3). To replace this sum with a maximum, we define a *max-product* $\otimes$ of matrices as follows.

Definition 4. *Let* $\mathbf{M}$ *and* $\mathbf{N}$ *be matrices of dimension* $m \times n$ *and* $n \times p$*, respectively. The* max-product *of* $\mathbf{M}$ *and* $\mathbf{N}$*, written* $\mathbf{M} \otimes \mathbf{N}$*, is a matrix of dimension* $m \times p$ *whose element at position* i *and* j *is equal to*

$$\max_{1 \leq k \leq n} a_{i,k} \cdot b_{k,j}.$$

It is straightforward to see that the max-product has the same properties as the standard matrix product, such as associativity and distributivity with respect to the sum. However, sum and maximum do not commute, so we cannot just replace $\times$ with $\otimes$ in equation (3) and hope to reformulate the model's equations analogously to Proposition 1. To facilitate such a reformulation, we also replace alls sums in equations (3) and (4) with maximums too. Thus, for matrices $\mathbf{M}$ and $\mathbf{N}$ of equal dimensions, we let $\max\{\mathbf{M}, \mathbf{N}\}$ be the matrix obtained by taking the element-wise maximum—that is, its element at position i and j is equal to $\max\{m_{i,j}, n_{i,j}\}$. We are now ready to define our max-Neural-LP model.

Definition 5. *A* max-Neural-LP *model is defined as the standard Neural-LP model, but equations* (3) *and* (4) *are replaced with equations* (16) *and* (17)*, respectively.*

$$\mathbf{u}^{\ell} = \max_{1 \leq k \leq \delta} \max_{0 \leq \ell' < \ell} a_{h,k,\ell} \cdot b_{h,\ell,\ell'+1} \cdot \mathbf{M}_k \otimes (\mathbf{u}^{\ell'})^{\intercal} \quad (16)$$

$$\mathbf{u}^{L+1} = \max_{0 \leq \ell \leq L} b_{h,L+1,\ell+1} \cdot \mathbf{u}^{\ell} \quad (17)$$

Just like a standard Neural-LP model, each max-Neural-LP model $\mathcal{M}$ defines an operator $T_{\mathcal{M}}$ on datasets, so we can straightforwardly apply the notions of soundness, completeness, and faithfulness to max-Neural-LP models. Moreover, a max-Neural-LP model is regular if all of its components are nonnegative. We are now ready to show that Algorithm 1 produces programs that are both sound and complete for each max-Neural-LP model provided we use the same threshold for fact classification and rule extraction. Towards this goal, we first adapt Proposition 1 to max-Neural-LP.

Proposition 3. *Let* $\mathcal{M} = (\mathbf{A}, \mathbf{B}, \beta)$ *be a regular max-Neural-LP model, let* D *be a dataset, and let* $\mathbf{u}^0, \dots, \mathbf{u}^{L+1}$ *be the vectors computed by* (16) *and* (17) *when checking whether* $R_h(c_i, c_j)$ *should be added to* $T_{\mathcal{M}}(D)$*. Then,* (18) *and* (19) *hold for each* $0 \leq \ell \leq L$ *and each* $1 \leq p \leq \epsilon$*.*

$$u^{\ell}_p = \max_{\lambda \in \mathcal{L}_{\ell}} \max_{\kappa \in \mathcal{K}_{\lambda}} \varphi(h, \kappa, \lambda) \max_{\tau \in \mathcal{T}_{\lambda,p,j}} \psi(\kappa, \tau) \quad (18)$$

$$u^{L+1}_p = \max_{\lambda \in \mathcal{L}} \max_{\kappa \in \mathcal{K}_{\lambda}} \theta(h, \kappa, \lambda) \max_{\tau \in \mathcal{T}_{\lambda,p,j}} \psi(\kappa, \tau) \quad (19)$$

Proof. The following identities clearly hold for each set of numbers b_i, each set of numbers $c_{i,j}$, and each $\alpha \geq 0$:

$$\alpha \max_i b_i = \max_i \alpha \cdot b_i,$$
$$\max_i \max_j c_{i,j} = \max_j \max_i c_{i,j}.$$

Thus, since $\mathcal{M}$ is regular, equations (16) and (17) can be rewritten to (18) and (19) in the same way as in Figure 1, but while using $\max$ instead of $\sum$. □

Intuitively, Proposition 3 suggests that each rule corresponding to the unfolding of rules (5)–(7) is applied in a max-Neural-LP model in isolation, and the confidence of the resulting fact is the maximum derived by one unfolding. The rule application now matches the rule extraction procedure, so we can show that, for $\gamma = \beta$, Algorithm 1 produces a program that is faithful to the model.

Theorem 3. *For* $\mathcal{M} = (\mathbf{A}, \mathbf{B}, \beta)$ *a regular max-Neural-LP model, program* $\mathcal{R}^{\beta}_{\mathcal{M}}$ *is faithful to* $\mathcal{M}$*.*

Proof. The poof that $\mathcal{R}^{\beta}_{\mathcal{M}}$ is sound is analogous to the proof of the first claim of Theorem 1 so we omit it for the sake of brevity. To show that $\mathcal{R}^{\beta}_{\mathcal{M}}$ is complete for $\mathcal{M}$, consider an arbitrary dataset D and fact $R_h(c_i, c_j) \in T_{\mathcal{M}}(D)$, and let $\mathbf{u}^0, \dots, \mathbf{u}^{L+1}$ be the vectors computed by (16) and (17) when checking whether $R_h(c_i, c_j)$ should be added to $T_{\mathcal{M}}(D)$. Also, choose arbitrary sequences $\lambda \in \mathcal{L}$, $\kappa \in \mathcal{K}_{\lambda}$, and $\tau \in \mathcal{T}_{\lambda,i,j}$ that maximise $\theta(h, \kappa, \lambda) \cdot \psi(\kappa, \tau)$. Then, we have $u^{L+1}_i = \theta(h, \kappa, \lambda) \cdot \psi(\kappa, \tau) > \beta$, where the equality holds by Proposition 3, and the inequality holds because $R_h(c_i, c_j) \in T_{\mathcal{M}}(D)$. Now $\beta \geq 0$ ensures $\psi(\kappa, \tau) = 1$; but then, for $\kappa = [k_1, \dots, k_n]$ and $\tau = [t_0, \dots, t_n]$, we have

$$\{R_{k_n}(c_{t_n}, c_{t_{n-1}}), \dots, R_{k_1}(c_{t_1}, c_{t_0})\} \subseteq D. \quad (20)$$

Finally, we have $\theta(h, \kappa, \lambda) > \beta$, so Definition 1 ensures that $\mathcal{R}^{\beta}_{\mathcal{M}}$ contains the chain rule (2). This rule clearly derives $R_h(c_i, c_j)$ on the facts from equation (20), as required. □

5 Evaluation

To evaluate our approach, we ported the code of Neural-LP to TensorFlow 2.7.0 and extended it with the ability to learn max-Neural-LP models. To reduce memory consumption, Neural-LP uses a sparse representation of adjacency tensors, and and it relies on TensorFlow's highly optimised implementation of multiplication of a sparse matrix by a

dense one. To use the same approach in max-Neural-LP, we extended TensorFlow with a custom implementation of the max-product of a sparse matrix by a dense one. We ran our experiments on a laptop running macOS 12.2 and Python 3.8.5 with 8 GB of RAM and an Intel Core i5 2.30 GHz CPU. We evaluated the two versions of Neural-LP on several well-known benchmarks. All systems, scripts, and datasets used in the experiments are available online.[1]

5.1 Benchmarks and Training

We evaluate our models on knowledge graph (KG) completion tasks, where the aim is to extend an incomplete KG represented as a dataset D to its complete version D' by adding missing facts. When seen as a classification problem, the objective of KG completion is to learn a Boolean function that takes as input a dataset D and a fact α over a fixed set of binary predicates and returns `true` if and only if $\alpha \in D'$.

We used the 12 benchmark datasets by Teru, Denis, and Hamilton (2020) based on FB15K-237 by Bordes et al. (2013), NELL-995 by Xiong, Hoang, and Wang (2017), and WN18RR by Dettmers et al. (2018). Each benchmark provides disjoint datasets $\mathcal{T}$, $\mathcal{V}$, and $\mathcal{S}$ for training, validation, and testing, respectively; relevant statistics are shown in Table 1. The benchmarks also provide means for splitting the test dataset $\mathcal{S}$ into disjoint subsets $\mathcal{S}_I$ and $\mathcal{S}_M$, where the former plays the role of the incomplete test KG and the latter contains the missing facts.

We trained both versions of Neural-LP as denoising autoencoders (Vincent et al. 2010). We split the training dataset $\mathcal{T}$ with a 3:1 ratio into an incomplete training KG $\mathcal{T}_I$ and a set $\mathcal{T}_M$ of missing facts, and we used $(\mathcal{T}_I, \alpha)$ for each fact $\alpha \in \mathcal{T}_M$ as a positive example. We trained models of depth $L = 3$ using Adam optimisation with the standard learning rate 0.001, a batch size of 64, and a maximum of 10 epochs. Following Yang, Yang, and Cohen (2017) in their evaluation of Neural-LP, we used log-likelihood loss with sum reduction, minimum probability threshold of 10^{-20}, and target value 1. We trained both versions of Neural-LP using an RNN neural controller system by Yang, Yang, and Cohen (2017). For each system and benchmark, the training process yields a pair of tensors $\mathbf{A}$ and $\mathbf{B}$ whose elements are computed using the softmax function and are thus between zero and one. We then combine these tensors with different nonnegative classification thresholds β to obtain models $\mathcal{M} = (\mathbf{A}, \mathbf{B}, \beta)$ that are regular according to Definition 2.

5.2 Rule Extraction Completeness for Neural-LP

As discussed in Section 3.3, Algorithm 1 produces sound programs for Neural-LP models if the rule extraction threshold γ is greater or equal to the model's classification threshold β. Using $\gamma = \beta$ clearly produces the largest such program, and one may wonder whether this program is 'sufficiently complete' in practice.

To answer this question, for each benchmark, we trained tensors $\mathbf{A}$ and $\mathbf{B}$ as described. Then, for each value of β shown in Table 2, we used Neural-LP's implementation of Algorithm 1 to extract the program $\mathcal{R}^{\beta}_{\mathcal{M}}$ from the model $\mathcal{M} = (\mathbf{A}, \mathbf{B}, \beta)$; we computed the sets of facts $T_{\mathcal{R}^{\beta}_{\mathcal{M}}}(\mathcal{S}_I)$ and $T_{\mathcal{M}}(\mathcal{S}_I)$ that the program and the model produce on the incomplete test dataset $\mathcal{S}_I$; and we calculated the percentage ratio of the sizes of these two sets. This provided us with a 'measure of completeness' of the extracted programs on the test datasets. Our results are shown in Table 2.

For benchmarks based on FB15K and NELL-995, the extracted rules often derive less than 3% of the facts predicted by the model; for benchmarks based on WN18RR, the rules typically derive 5%–30% of the predicted facts. We take this as empirical evidence that the extracted rules do not sufficiently explain the predictions of Neural-LP models. We observe also that the extracted rules provide a better approximation for smaller thresholds. This is because the number of extracted rules decreases quickly when γ rises to 0.01 and above: for models of depth 3, the values $\theta(h, \kappa, \lambda)$ are a product of 7 elements of $\mathbf{A}$ and $\mathbf{B}$; now these elements are typically smaller than 0.5, and $0.5^7 \approx 0.008$.

5.3 Knowledge Graph Completion Performance

We next compared the performance of Neural-LP and max-Neural-LP models on KG completion. For each benchmark, we proceeded as follows. For each fact $\alpha \in \mathcal{S}_M$, we took $(\mathcal{S}_I, \alpha)$ as a positive testing example. We generated negative testing examples by uniformly sampling at random $|\mathcal{S}_M|$ facts from the set of all facts $R(a, b) \notin \mathcal{S}$ where predicate R is used in the benchmark, and constants a and b are at distance at most 3 in $\mathcal{S}_I$—that is, either $a = b$, or there exist facts $\{R_1(a, c_1), \ldots, R_n(c_{n-1}, b)\} \subseteq \mathcal{S}_I$ for $n \in \{1, 2, 3\}$. We used this negative sampling strategy because a model's depth fundamentally limits the type of inferences the model can make: a fact of the form $R(a, b)$ can be derived on a dataset D only if the distance of constants a and b in D is at most the model's depth. For each system and benchmark, we computed the F1 score on the validation dataset for a range of classification thresholds β between 0 and 1, and we selected the threshold that maximised this score. For Neural-LP models, this threshold was between 0.001 and 1 for most benchmarks, whereas for max-Neural-LP models, the threshold was typically around 10^{-5}. For the selected threshold, we classified the positive and negative testing examples, and we computed precision, recall, accuracy, and F1 score as usual. Finally, we computed the area under the precision-recall curve (AUC) using the thresholds considered during validation. Our results are shown in Table 3.

The performance of max-Neural-LP models is on a par with that of Neural-LP models. Recall values of both systems are generally low in all benchmarks; this can be explained by the fact that many positive examples involve constants separated by more than three steps in $\mathcal{S}_I$, so they cannot be predicted by either approach. The benchmarks based on WN18RR were difficult for both systems: the systems produced many false positives, which led to lower precision scores. Finally, the training times of Neural-LP and max-Neural-LP models were comparable.

Our results confirm that max-Neural-LP models can be effectively trained in practice to achieve performance comparable to Neural-LP models. Yang, Yang, and Cohen (2017)

[1] http://krr-nas.cs.ox.ac.uk/2022/max-Neural-LP/

	FB15K-237				NELL-995				WN18RR			
	v1	v2	v3	v4	v1	v2	v3	v4	v1	v2	v3	v4
# Facts for Training	4,245	9,739	17,986	27,203	4,687	8,219	16,393	7,546	5,410	15,262	25,901	7,940
# Facts for Validation	489	1,166	2,194	3,352	414	922	1,851	876	630	1,838	3,097	934
# Facts for Testing	2,198	4,623	8,271	13,138	933	5,062	8,857	7,804	1,806	4,452	6,932	13,763
# Predicates	180	200	215	219	14	88	142	76	9	10	11	9

Table 1: Numbers of Predicates and Facts in Training, Validation, and Testing Datasets Per Benchmark

	GraIL-BM/FB15K-237				GraIL-BM/NELL-995				GraIL-BM/WN18RR			
Threshold β	v1	v2	v3	v4	v1	v2	v3	v4	v1	v2	v3	v4
0.001	4.2	2.4	1.1	0.97	29	3.2	1.3	1.6	43	37	9.9	54
0.002	1.5	1.1	0.62	0.55	25	2.0	0.63	1.7	35	27	6.0	39
0.005	0.29	0.28	0.31	0.24	3.9	0.76	0.33	0.52	26	17	11	25
0.01	0.089	0.13	0.21	0.20	4.9	0.50	0.28	0.23	20	11.5	8.2	20
0.02	0.056	0.15	0.14	0.19	3.1	0.43	0.17	0.14	17	7.9	6.6	12
0.05	0.054	0.10	0.14	0.21	1.1	0.35	0.13	0.22	7.9	5.6	3.2	7.9
0.1	0.051	0.089	0.086	0.14	0.58	0.18	0.051	0.24	6.1	3.3	3.2	5.6
0.2	0.044	0.0	0.0	0.0075	0.18	0.098	0.0	0.51	0.98	4.4	3.9	5.1
0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 2: Percentage of Facts Derived by Neural-LP Model $\mathcal{M}$ for Threshold β Also Derived by $\mathcal{R}^{\beta}_{\mathcal{M}}$

		GraIL-BM/FB15K-237				GraIL-BM/NELL-995				GraIL-BM/WN18RR			
	Model	v1	v2	v3	v4	v1	v2	v3	v4	v1	v2	v3	v4
Precision	Neural-LP	**37.8**	45.4	67.1	64.6	38.5	69.8	76.8	59.5	19.7	**18.2**	3.8	16.1
	Max-Neural-LP	30.7	**75.1**	**74.7**	**85.4**	**39.5**	**80.1**	**77.1**	**70.4**	**20.5**	12.5	**5.8**	**16.9**
Recall	Neural-LP	16.6	**34.1**	**25.7**	**33.7**	60.0	26.7	30.3	6.8	**13.8**	0.9	0.3	**11.1**
	Max-Neural-LP	**19.0**	31.0	25.5	30.4	**62.0**	**35.9**	**40.8**	**53.4**	13.3	**7.7**	**4.3**	11.0
Accuracy	Neural-LP	**44.6**	46.5	56.5	57.6	32.0	57.6	60.6	51.1	28.7	**48.4**	**46.0**	26.7
	Max-Neural-LP	38.0	**60.4**	**58.4**	**62.6**	**33.5**	**63.7**	**64.3**	**65.5**	**30.9**	26.8	17.0	**28.5**
F1 Score	Neural-LP	23.1	38.9	37.1	44.3	46.9	38.6	43.4	12.3	**16.3**	1.7	0.6	13.1
	Max-Neural-LP	**23.5**	**43.9**	**38.1**	**44.8**	**47.3**	**49.7**	**64.3**	**60.7**	16.1	**9.5**	**4.9**	**13.3**
AUC	Neural-LP	23.3	39.3	40.7	48.5	20.6	45.4	50.1	48.9	**14.0**	**9.6**	**3.1**	**9.7**
	Max-Neural-LP	**25.4**	**44.8**	**44.1**	**50.4**	**56.7**	**55.1**	**53.4**	**56.7**	11.5	8.1	2.8	8.3
Training Time	Neural-LP	3.8	**7.5**	46.2	90.7	0.4	5.7	35.8	**2.4**	0.4	2.8	**4.8**	1.6
	Max-Neural-LP	**3.3**	14.3	**39.4**	**85.9**	0.4	**5.2**	**32.3**	4.5	**0.3**	**2.7**	9.5	1.6

Table 3: Classification Metrics (%) and Training Times (minutes) for Both Systems

compare Neural-LP with other rule learning and rule mining approaches, and their results transfer to max-Neural-LP as well. The crucial benefit of max-Neural-LP models is that we can extract from each model a program that faithfully captures the model's predictions on any dataset.

6 Conclusion and Future Work

In this paper, we studied the formal properties of the Neural-LP approach to rule learning. We showed that there is a fundamental mismatch between the predictions made by Neural-LP models and the inferences of the rules extracted from such models. We proposed a novel family of max-Neural-LP models for which the extracted Datalog rules completely characterise the models' predictions. Our max-Neural-LP models achieve comparable performance to that of Neural-LP on knowledge graph completion benchmarks. Thus, we show that it is possible to devise practical rule learning approaches with a formally well-understood relationship between the model and the extracted rules.

For our future work, we shall extend our results to approaches that build on top of Neural-LP such as those by Sadeghian et al. (2019) and Wang et al. (2020), as well as approaches based on different types of models, such as those by Evans and Grefenstette (2018) and Qu et al. (2021).

Acknowledgments

This work was supported by the AIDA project (Alan Turing Institute, EP/N510129/1), the SIRIUS Centre for Scalable Data Access (Research Council of Norway, project number 237889), Samsung Research UK, Siemens AG, and the EPSRC projects ConCur (EP/V050869/1), OASIS (EP/S032347/1) and UK FIRES (EP/S019111/1).

References

Abiteboul, S.; Hull, R.; and Vianu, V. 1995. *Foundations of Databases*. Addison Wesley.

Ahmadi, N.; Huynh, V.-P.; Meduri, V. V.; Ortona, S.; and Papotti, P. 2020. Mining Expressive Rules in Knowledge Graphs. *Journal of Data and Information Quality* 12(2):8:1–8:27.

Arni, F.; Ong, K.; Tsur, S.; Wang, H.; and Zaniolo, C. 2003. The deductive database system LDL++. *TPLP* 3(1):61–94.

Bader, S.; Hitzler, P.; Hölldobler, S.; and Witzel, A. 2007. A Fully Connectionist Model Generator for Covered First-Order Logic Programs. In Veloso, M. M., ed., *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007)*, 666–671.

Bader, S.; d'Avila Garcez, A. S.; and Hitzler, P. 2005. Computing First-Order Logic Programs by Fibring Artificial Neural Networks. In Russell, I., and Markov, Z., eds., *Proc. of the 18th Int. Florida Artificial Intelligence Research Society Conference (FLAIRS 2005)*, 314–319. AAAI Press.

Bordes, A.; Usunier, N.; García-Durán, A.; Weston, J.; and Yakhnenko, O. 2013. Translating embeddings for modeling multi-relational data. In Burges, C. J. C.; Bottou, L.; Ghahramani, Z.; and Weinberger, K. Q., eds., *NeurIPS*, 2787–2795.

Campero, A.; Pareja, A.; Klinger, T.; Tenenbaum, J.; and Riedel, S. 2018. Logical rule induction and theory learning using neural theorem proving. *CoRR* abs/1809.02193.

Cropper, A.; Dumancic, S.; Evans, R.; and Muggleton, S. H. 2021. Inductive logic programming at 30. *CoRR* abs/2102.10556.

Dettmers, T.; Minervini, P.; Stenetorp, P.; and Riedel, S. 2018. Convolutional 2D knowledge graph embeddings. In McIlraith, S. A., and Weinberger, K. Q., eds., *AAAI*, 1811–1818. AAAI Press.

Dong, H.; Mao, J.; Lin, T.; Wang, C.; Li, L.; and Zhou, D. 2019. Neural Logic Machines (Poster). In *Proc. of the 7th Int. Conf. on Learning Representations (ICLR 2019)*.

Eisner, J., and Filardo, N. W. 2010. Dyna: Extending Datalog for Modern AI. In *Datalog Reloaded—First International Workshop, Datalog 2010. Revised Selected Papers*, 181–220. Oxford, UK: Springer.

Evans, R., and Grefenstette, E. 2018. Learning Explanatory Rules from Noisy Data. *Journal of Artificial Intelligence Research* 61:1–64.

Galárraga, L.; Teflioudi, C.; Hose, K.; and Suchanek, F. M. 2015. Fast Rule Mining in Ontological Knowledge Bases with AMIE+. *VLDB J.* 24(6):707–730.

Green, T. J.; Huang, S. S.; Loo, B. T.; and Zhou, W. 2013. Datalog and Recursive Query Processing. *Foundations and Trends in Databases* 5(2):105–195.

Hölldobler, S.; Kalinke, Y.; and Störr, H.-P. 1999. Approximating the Semantics of Logic Programs by Recurrent Neural Networks. *Applied Intelligence* 11(1):45–58.

Meilicke, C.; Chekol, M. W.; Ruffinelli, D.; and Stuckenschmidt, H. 2019. Anytime Bottom-Up Rule Learning for Knowledge Graph Completion. In Kraus, S., ed., *Proc. of the 28th Int. Joint Conf. on Artificial Intelligence (IJCAI 2019*, 3137–3143.

Muggleton, S. 1991. Inductive Logic Programming. *New Generation Computing* 8(4):295–318.

Qu, M.; Chen, J.; Xhonneux, L.-P. A. C.; Bengio, Y.; and Tang, J. 2021. RNNLogic: Learning Logic Rules for Reasoning on Knowledge Graphs. In *Proc. of the 9th Int. Conf. on Learning Representations (ICLR 2021)*.

Rocktäschel, T., and Riedel, S. 2017. End-to-end Differentiable Proving. In Guyon, I.; von Luxburg, U.; Bengio, S.; Wallach, H. M.; Fergus, R.; Vishwanathan, S. V. N.; and Garnett, R., eds., *Proc. of the 31st Conf. on Neural Information Processing Systems (NeurIPS 2017)*, 3788–3800.

Rossi, A.; Barbosa, D.; Firmani, D.; Matinata, A.; and Merialdo, P. 2021. Knowledge graph embedding for link prediction: A comparative analysis. *ACM Trans. Knowl. Discov. Data* 15(2):14:1–14:49.

Sadeghian, A.; Armandpour, M.; Ding, P.; and Wang, D. Z. 2019. DRUM: End-To-End Differentiable Rule Mining On Knowledge Graphs. In Wallach, H. M.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E. B.; and Garnett, R., eds., *Proc. of the 33rd Conf. on Neural Information Processing Systems (NeurIPS 2017)*, 15321–15331.

Teru, K. K.; Denis, E.; and Hamilton, W. 2020. Inductive relation prediction by subgraph reasoning. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, 9448–9457. PMLR.

Vincent, P.; Larochelle, H.; Lajoie, I.; Bengio, Y.; Manzagol, P.-A.; and Bottou, L. 2010. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. of Machine Learning Res.* 11(12).

Wang, P.-W.; Stepanova, D.; Domokos, C.; and Kolter, J. Z. 2020. Differentiable learning of numerical rules in knowledge graphs. In *Proc. of the 8th Int. Conf. on Learning Representations (ICLR 2020)*.

Xiong, W.; Hoang, T.; and Wang, W. Y. 2017. Deeppath: A reinforcement learning method for knowledge graph reasoning. In *EMNLP*, 564–573. Association for Computational Linguistics.

Yang, F.; Yang, Z.; and Cohen, W. W. 2017. Differentiable Learning of Logical Rules for Knowledge Base Reasoning. In Guyon, I.; von Luxburg, U.; Bengio, S.; Wallach, H. M.; Fergus, R.; Vishwanathan, S. V. N.; and Garnett, R., eds., *Proc. of the 31st Conf. on Neural Information Processing Systems (NeurIPS 2017)*, 2319–2328.